I-right/TK取扱説明書

μTeaboard/ARM7-AT91用

Version 1.10

パーソナルメディア株式会社

目次

修正履歴

改版	摘要
1.10	移行ガイド (ITRON から T-Kernel への移行) を追加
1.00	新規作成

1 はじめに

1.1 I-right/TK の概要

「I-right/TK」は、ITRON 用に作成されたプログラムを T-Kernel 上で実行するためのラッパー (wrapper) です。主な特長は次のとおりです。

- **ITRON の資産を T-Kernel で活用**

  これまで組込み向けに開発されてきた ITRON の豊富なプログラム資産を、最小限の修正により T-Engine プロジェクトの成果である T-Kernel 上で実行できます。また ITRON に慣れたエンジニアが、T-Kernel 上で新規のプログラムを開発する場合にも便利です。

- **μITRON4.0 フルセットに準拠**

  ITRON の API を T-Kernel ベースのプログラム上から呼び出せます。ITRON API の仕様は μITRON4.0 仕様のフルセットにほぼ準拠しています。

- **固定的なオブジェクト ID 番号の指定や静的 API にも対応**

  T-Kernel ではタスク ID などのオブジェクト ID 番号は動的に決まるため、固定値では指定できませんが、I-right/TK では ITRON と同様に、固定的なオブジェクト ID 番号で指定可能です。また、μITRON 4.0 仕様の静的 API にも対応しています。

- **軽量なラッパー**

  I-right/TK は、μITRON 4.0 と T-Kernel の仕様の差の吸収したり、ITRON のオブジェクト ID (固定 ID) と T-Kernel のオブジェクト ID (動的 ID) の変換を行う軽量なラッパーです。この方式の採用により、ITRON と T-Kernel の持つリアルタイム性をどちらも損なうことなく、ITRON と T-Kernel の連携動作が可能になります。

- **T-Kernel のミドルウェアやアプリケーションとの連携**

  ITRON 側から T-Kernel 上の豊富なデバイスドライバやミドルウェアを利用したり、T-Kernel 上のプログラムと連携動作を行うことが可能です。例えば T-Kernel 上で動いているファイルシステムや TCP/IP と、既存の ITRON のプログラムを組合せたシステムを容易に構築できます。

- **多くの 32 ビット CPU をサポート**

  T-Kernel は現在市販されている多くの 32 ビット CPU に対応済みであり、その成果が ITRON のプログラムからも利用できます。

- **Eclipse による開発**

  Eclipse によるプログラム開発が可能です。

1.2 リリース内容

I-right/TK のリリースには以下の内容が含まれています。

ドキュメント

- 『I-right/TK 取扱説明書』

  本書です。インストール方法や API の仕様などを含むマニュアルです。

- 『ITRON から T-Kernel への移行』

  ITRON のプログラムを T-Kernel に移行するためのガイドです。

ソフトウェア

- インストール用アーカイブ：`irtk.tbat91.02.zip`

  I-right/TK のインストール用アーカイブです。開発ホストとなるパソコンの開発環境上に展開します。

  † 上記のアーカイブファイル名の数字は、リリースのバージョンによって異なる場合があります。

- サンプルプログラム集：`sample.tbat91.02.zip`

  I-right/TK を使ってプログラムを作成する上で参考となるサンプルプログラム集です。下記のすべてのサンプルがアーカイブされています。展開してご利用ください。

`sample_first`	インストール後の動作確認用の簡単なサンプル
`sample_itron`	ITRON の各機能を使うサンプル
`sample_isr`	割込みサービスルーチンを使うサンプル
`sample_lib`	ライブラリを使うサンプル
`sample_net`	ネットワーク (TCP/IP) にアクセスするサンプル
`sample_dev`	デバイスドライバを呼び出すサンプル
`sample_sdi`	デバイスドライバを作成して他のプログラムと連携動作するサンプル

2 セットアップ

2.1 開発環境のセットアップ

I-right/TK のインストールに先立って、開発ホストとなるパソコン側と μTeaboard 側のセットアップを行い、T-Kernel ベースのプログラムを作成、実行できる環境を整える必要があります。詳細につきましては μTeaboard/ARM7-AT91 のドキュメントをご参照ください。

ドキュメント『はじめてみよう μTeaboard』に沿って最初から順にインストールしていただき、「Hello, world (T-Kernel ベース編)」までをお試しいただくことをおすすめします。

開発環境のベースディレクトリ ($BD)

この文書では、開発ホストとなるパソコンの μTeaboard/ARM7-AT91 用開発環境のベースディレクトリを「$BD」で表します。

Eclipse 版開発環境の場合:
$BD は標準では次のとおりです。

```
C:¥eclipse¥plugins¥com.t_engine4u.tl.tbat91.1.1.0_1.1.0¥te
```

† 上記のパス名は開発環境のインストール先やバージョンによって異なる場合があります。

Cygwin 上のコンソール版の開発環境の場合:
$BD は標準では次のとおりです。

/usr/local/te
(Windows から見た場合は C:¥cygwin¥usr¥local¥te)

† 上記のパス名は Cygwin や開発環境のインストール先によって異なる場合があります。

Linux 上のコンソール版の開発環境の場合:
$BD は標準では次のとおりです。

/usr/local/te

† 上記のパス名は開発環境のインストール先によって異なる場合があります。

2.2　I-right/TK のインストール

インストール用アーカイブ irtk.tbat91.02.zip を、$BD に展開してください。

† 上記のアーカイブファイル名の数字は、リリースのバージョンによって異なる場合があります。

展開すると、次のファイルが作成されます。

$BD/include/itron/itron.h　— ヘッダファイル
$BD/lib/tbat91/libitron.a　— ライブラリ

2.3　I-right/TK の動作確認

I-right/TK のインストールが正常に行われたことを確認するため、ITRON の簡単なプログラムをメイクして実行してみましょう。ここでは I-right/TK に付属するサンプル「sample_first」をメイクして実行します。

Eclipse 版開発環境の場合

(1) **Eclipse の起動**

開発ホストとなるパソコン上で Eclipse のアイコンをダブルクリックして起動します。冒頭のワークスペースの選択では、T-Kernel ベース用の標準のワークスペースである「C:¥te¥kappl」を選択してください。

(2) **プロジェクトの作成**

Eclipse の T-Engine 開発パースペクティブで、次のようにプロジェクトを新規作成します。

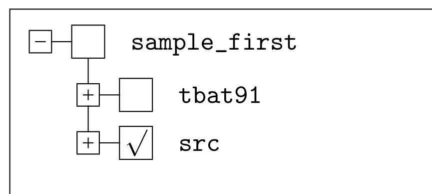
- メニューバーの「ファイル」→「新規」→「T-Engine C/C++ プロジェクト」を選択して、「新規プロジェクト」ダイアログを開きます。
- 「プロジェクト名」は「sample_first」としてください。
- 「ターゲット」は対象機種 (tbat91) を選択してください。
- 「プログラムタイプ」は「T-Kernel Base」を選択してください。ワークスペースを「C:¥te¥kappl」としていれば、「ワークスペース名による自動決定」のままでも自動的に T-Kernel ベースになります。
- 「テンプレートプログラムの生成」はチェックしないでください。
- 「出力ディレクトリの生成」はチェックしてください。

- 最後に **完了** をクリックすると、プロジェクトが生成されます。

(3) ソースのインポート

「C/C++ プロジェクト」内に作成されたプロジェクト「sample_first」を選択して、次のようにソースをインポートします。

- メニューバーの「ファイル」→「インポート」を選択して、「インポート」ダイアログを開きます。
- 「一般」の左の [+] をクリックして開き、「ファイルシステム」を選択して **次へ** をクリックします。
- ソースディレクトリ欄の右の **参照** をクリックして「ディレクトリからインポート」ダイアログを開き、インポートしたいプログラム全体のフォルダを選択して、**開く** をクリックします。今回は I-right/TK に付属するサンプルプログラムのアーカイブを展開して、展開された中の「sample_first」を選択してください。
- インポートする対象として、sample_first の左の [+] をクリックして開き、src ディレクトリのみをチェックして、**完了** をクリックします。



(4) メイク

プロジェクト内のオブジェクト出力ディレクトリ (tbat91) を選択した状態で、メニューバーの「プロジェクト」→「T-Engine Target の Make all」を選択してメイクします。

プロジェクト内のオブジェクト出力ディレクトリ (tbat91) を開いて、次の実行ファイルが生成されたことを確認します。

- sample_first.abs
  絶対アドレス形式の実行ファイルです。
- sample_first.mot
  上記の実行ファイルを S レコード形式に変換したものです。ファイル転送用です。

(5) ファイル転送とロード・実行

- あらかじめ Eclipse の外部ツール (メニューバーの「実行」→「外部ツール」) の te_vcom と gterm を起動しておきます。gterm コンソール上で Enter キーを何回か押して、ターゲット (μTeaboard) 側から IMS のプロンプト [IMS]% が返ってくることを確認します。
- オブジェクト出力ディレクトリ (tbat91) にメイクされた実行ファイル sample_first.abs を選択して、右クリックメニューの「実行」→「構成および実行」を選択して、「構成および実行」ダイアログを開きます。

  「T-Engine アプリケーション」の上で右クリックして「新規」を選択します。「sample_first」の転送方法や実行方法などの設定が自動的に行われます。

  「メイン」タブの「転送ファイル」は空欄になっていますが、転送用ファイルとして「tbat91¥sample_first.mot」を指定する必要があります。

  実行 をクリックすると、自動的に .load コマンドによるファイル転送と lodspg コマンドによるロードが行われ、実行が開始されます。
- このプログラムはタスクと周期ハンドラを利用して、「Hello, world」というメッセージをコンソール上に 1 秒ごとに合計 5 回出力し、最後に「Good bye」と出力して終了します。

(6) **アンロード**

このプログラムの実行が終わっても、プログラムそのものはまだメモリ上 (システム共有空間) にロードされた状態で残っています。これは次のようにアンロード可能です。

```
[IMS]% unlspg @0x20100000 ↩
```

Cygwin 上または Linux 上のコンソール版の開発環境の場合

(1) **ソースの展開**

I-right/TK に付属するサンプル「sample_first」を、$BD/kappl 以下にディレクトリごとコピーします。

(2) **メイク**

オブジェクト出力ディレクトリ (tbat91) 上でメイクします。

```
$ cd $BD/kappl/sample_first/tbat91 ↩
$ make ↩
```

次の実行ファイルが生成されたことを確認します。

- sample_first.abs
  絶対アドレス形式の実行ファイルです。
- sample_first.mot
  上記の実行ファイルを S レコード形式に変換したものです。ファイル転送用です。

(3) ファイル転送とロード・実行

オブジェクト出力ディレクトリ (tbat91) の下にメイクされた S レコードファイル sample_first.mot を、T-Monitor 上の load コマンド (端末ソフトとして gterm を使用している場合は .load コマンド) で実機のメモリ上に転送して、IMS の lodspg コマンドで実行します。

```
[IMS]% #↩
TM> load xs↩   (gterm 以外の場合は、XMODEM で
  $BD/kappl/sample_first/tbat91/
  の下の sample_first.mot を送信してください)
または TM> .load sample_first.mot↩   (gterm の場合)
TM> g↩
[IMS]% lodspg @0x20100000↩
```

このプログラムはタスクと周期ハンドラを利用して、「Hello, world」というメッセージをコンソール上に 1 秒ごとに合計 5 回出力し、最後に「Good bye」と出力して終了します。

(4) アンロード

このプログラムの実行が終わっても、プログラムそのものはまだメモリ上 (システム共有空間) にロードされた状態で残っています。これは次のようにアンロード可能です。

```
[IMS]% unlspg @0x20100000↩
```


3 ITRON ラッパー機能の詳細

3.1 ITRON 機能の利用方法

I-right/TK の ITRON の機能を利用してプログラムを作成する方法を説明します。

なお、ITRON の機能を使った具体的なプログラム例としては、以下のサンプルが付属していますので、あわせてご参照いただければ幸いです。

sample_first	タスクと周期ハンドラを使った簡単なサンプル
sample_itron	ITRON の各機能を使うサンプル
sample_isr	割込みサービスルーチンを使うサンプル

3.1.1 プログラムのタイプ

I-right/TK の ITRON の機能は、T-Kernel ベースのプログラムから利用することができます。

Eclipse 版開発環境をご利用の場合は、作成するプログラムのプロジェクトの生成時にプログラムタイプとして T-Kernel ベースを指定してください。

コンソール版の開発環境をご利用の場合は、作成するプログラムの Makefile 内で、T-Kernel ベースの標準ルール
`$(BD)/kappl/etc/makerules` をインクルードしてください。

3.1.2 I-right/TK のライブラリのリンク指定

プログラムのメイク時に I-right/TK のライブラリ `libitron.a` をリンクする必要があります。このため Makefile 内のオプション設定のところに、次のようにライブラリの指定を追加してください。

```
# オプション
CFLAGS += $(CFLAGS_WARNING)
LOADLIBES += -litron
```

† **複数のプログラムからの ITRON 機能の利用について**

ITRON のプログラムが複数に分かれている場合でも、基本的には、一つにまとめた上で I-right/TK のライブラリと一括リンクするようにしてください。

複数のプログラムを一つにまとめず別々にリンクして、それぞれのプログラムから ITRON 機能を使うことも可能ですが、その場合は個々のリンク単位で ITRON 機能が別々に動作する形になります。特に ITRON のタスク ID などのオブジェクト ID は、ID 番号が同じでも実際には別々のオブジェクトを表します。

3.1.3 ヘッダーファイルのインクルード

プログラムのソースは C 言語で記述します。ソースの冒頭で、次のようにヘッダーファイルのインクルードすることにより、ITRON の各 API が使えるようになります。

```
#include <itron/itron.h> /* ITRON ヘッダー */
```

他のヘッダーファイルをインクルードする場合は、`itron/itron.h` より後でインクルードしてください。`itron/itron.h` より先に他のヘッダーファイルをインクルードすると、問題が生じる可能性があります。

3.1.4 I-right/TK の初期化とコンフィギュレーション

ITRON のタスクやハンドラの起動に先立って、I-right/TK のシステムを初期化しておく必要があります。この処理を「コンフィギュレーション」と呼びます。大文字表記の静的 API を使ってあらかじめオブジェクトを生成しておく場合には、コンフィギュレーションの中でその処理を行います。

ITRON の機能を使うプログラムは、必ず T-Kernel ベースのプログラムとして作成する必要があります。そのプログラムのロード時に実行される `main` 関数を以下のように記述しておくことにより、コンフィギュレーションを行います。

```
#include <itron/itron.h> /* ITRON ヘッダー */

/* ITRON コンフィギュレーション */
ER main( INT ac, UB *av[] )
{
	if (ac < 0) return CLR_CFG; /* アンロード処理 */

	/* ロード処理 */
	STA_CFG; /* コンフィギュレーション開始 */
	··· CRE_TSK などの静的 API ···
	END_CFG; /* コンフィギュレーション終了 */
	return ERR_CFG; /* エラーが発生した場合はロードしない */
}
```

- `STA_CFG` と `END_CFG` の間には、次の静的 API を記述できます。

CRE_TSK	タスクの生成
DEF_TEX	タスク例外処理ルーチンの定義
CRE_SEM	セマフォの生成
CRE_FLG	イベントフラグの生成
CRE_DTQ	データキューの生成
CRE_MBX	メールボックスの生成
CRE_MTX	ミューテックスの生成
CRE_MBF	メッセージバッファの生成
CRE_POR	ランデブポートの生成
CRE_MPF	固定長メモリプールの生成
CRE_MPL	可変長メモリプールの生成
CRE_CYC	周期ハンドラの生成
CRE_ALM	アラームハンドラの生成
DEF_OVR	オーバーランハンドラの定義
DEF_INH	割込みハンドラの定義
ATT_ISR	割込みサービスルーチンの追加
ATT_INI	初期化ルーチンの追加

- lodspg コマンドによるプログラムのロード時 (ac >= 0 の場合) には、コンフィギュレーションを STA_CFG で開始し、END_CFG で終了してください。なお静的 API を一つも使わない場合でも、I-right/TK の初期化のためにコンフィギュレーションは必須ですので、STA_CFG と END_CFG を必ず呼び出してください。

- STA_CFG と END_CFG の間はディスパッチと割込みを禁止した状態で実行されます。このため CRE_TSK で TA_ACT 属性をつけてタスクを生成・起動した場合でも、END_CFG を呼び出すまでそのタスクの実行は遅延されます。また DEF_INH や ATT_ISR で割込みハンドラや割込みサービスルーチンを定義した場合でも、END_CFG を呼び出すまでは割込みは発生しません。

  ただし CRE_MPF (固定長メモリプール生成) と CRE_MPL (可変長メモリプール生成) を実行する時のみ、コンフィギュレーション中でもディスパッチと割込みが一時的に許可されます。この影響を避けるには、CRE_MPF や CRE_MPL を使う場合は STA_CFG の直後に記述してください。

  また CRE_CYC (周期ハンドラ生成) で属性に TA_STA を指定し、さらに起動位相に 0 を指定した場合は、コンフィギュレーション中でもただちに周期ハンドラが呼ばれます。この影響を避けるには、CRE_CYC では TA_STA を指定せず、コンフィギュレーション後に sta_cyc で周期ハンドラをスタートさせてください。

- コンフィギュレーションが正常に終了した場合は、ERR_CFG の値は 0 になっています。return ERR_CFG; とすることで、プログラムはメモリ空間上にロードされます。

  一方、コンフィギュレーション中にエラーが発生した場合は、ERR_CFG の値はマイナスになっています。return ERR_CFG; とすることで、プログラムはメモリ空間上にロードされず削除されます。このとき ERR_CFG の値の意味は、上位 16 ビットは最初に発生したエラーのエラーコード、下位 16 ビットはそのエラーの発生した行番号を表します。

  > (例) コンフィギュレーション中に誤って同一のタスク ID で複数のタスクを重ねて生成しようとした場合はエラーが発生し、return ERR_CFG; によって次のようなメッセージが表示され、プログラムはロードされません。
  >
  > ```
  > Can't load system program -2686929
  > ```
  >
  > ここで -2686929 (0xffd7002f) の上位 16 ビットの-41 (0xffd7) は E_OBJ エラー (対象オブジェクトが登録済み)、下位 16 ビットの 47 (0x002f) はエラーの発生した行番号を示します。

- unlspg コマンドによるプログラムのアンロード時 (ac < 0 の場合) には、アンロードに対応する場合は、CLR_CFG を呼び出して return で終了してください。これにより ITRON のオブジェクトが自動的に削除されます。

  ただし ITRON のプログラム内でメモリを確保したり、デバイスドライバを定義した場合など、ITRON のオブジェクト以外については自動的には削除されません。これらについてはアンロード処理の中で解放や削除を行ってください。

- なおアンロードに対応しなくてよい場合 (システムの電源を落としたりリセットするまでそのプログラムを使い続ける場合) は、アンロード時に CLR_CFG を呼び出さずに単に return でマイナスの値 (T-Kernel のエラーコード) を返してください。

  > (例) `if (ac < 0) return E_NOSPT << 16;`
  >
  > itron/itron.h で定義される ITRON のエラーコードを T-Kernel のエラーコードに変換するため 16 ビット左シフトします。

3.2 提供される API と制限事項

μITRON4.0 仕様の API のうち、I-right/TK でサポートする API と制限事項をこの節でまとめます。

† 可能な限り μITRON4.0 仕様に合わせていますが、T-Kernel との仕様の違いにより、一部に以下に示す制限事項があります。

なお、μITRON4.0 仕様の詳細については、トロンプロジェクトから公開されている仕様書をご参照ください。

3.2.1 タスク管理機能

ITRON の API 名称	機能	利用の可否と制限事項
CRE_TSK, cre_tsk, acre_tsk	タスクの生成	○ *1, 2
del_tsk	タスクの削除	○
act_tsk, iact_tsk	タスクの起動	○
can_act	タスクの起動要求のキャンセル	○
sta_tsk	タスクの起動 (起動コード指定)	○
ext_tsk	自タスクの終了	○
exd_tsk	自タスクの終了と削除	○
ter_tsk	タスクの強制終了	○
chg_pri	タスク優先度の変更	○
get_pri	タスク優先度の参照	○
ref_tsk, ref_tst	タスクの状態参照	○ *3

*1 スタック領域は常に OS 側で確保しますので、タスク生成時のスタック領域の先頭番地の指定 (stk) はサポートしません。常に NULL とみなされます。

*2 タスク生成時の属性として、TA_RNG0 (保護レベル 0) または TA_RNG1 (保護レベル 1) が論理和で追加指定できます。指定しない場合は保護レベル 0 となります。なお保護レベルの詳細については、『T-Kernel 仕様書』と μTeaboard/ARM7-AT91 の『実装仕様書』をご参照ください。

*3 lefttmo はサポートせず、常に 0 となります。またランデブ回送後で返答前の呼出しタスクの taskstat は TTW_CAL | TTW_RDV となり wobjid は 0 となります。

3.2.2 タスク付属同期機能

ITRON の API 名称	機能	利用の可否と制限事項
slp_tsk, tslp_tsk	起床待ち	○
wup_tsk, iwup_tsk	タスクの起床	○
can_wup	タスク起床要求のキャンセル	○
rel_wai, irel_wai	待ち状態の強制解除	○
sus_tsk	強制待ち状態への移行	○
rsm_tsk	強制待ち状態からの再開	○
frsm_tsk	強制待ち状態からの強制再開	○
dly_tsk	自タスクの遅延	○

3.2.3 タスク例外処理機能

ITRON の API 名称	機能	利用の可否と制限事項
DEF_TEX, def_tex	タスク例外処理ルーチンの定義	○ *4
ras_tex	タスク例外処理の要求	○
iras_tex	タスク例外処理の要求 (非タスク)	× *5
dis_tex	タスク例外処理の禁止	○
ena_tex	タスク例外処理の許可	○
sns_tex	タスク例外処理禁止状態の参照	○
ref_tex	タスク例外処理の状態参照	○

*4 タスク例外処理ルーチン属性は TA_HLNG のみ指定可能です。TA_ASM はサポートしません。また、タスク例外処理ルーチンを定義するタスクは保護レベル 1 (TA_RNG1) で生成する必要があります。

*5 タスク例外処理の要求は、タスクから ras_tex で行うようにしてください。非タスクコンテキストからの iras_tex はサポートしません。

3.2.4 セマフォ

ITRON の API 名称	機能	利用の可否と制限事項
CRE_SEM, cre_sem, acre_sem	セマフォの生成	○
del_sem	セマフォの削除	○
sig_sem, isig_sem	セマフォ資源の返却	○
wai_sem, pol_sem, twai_sem	セマフォ資源の獲得	○
ref_sem	セマフォの状態参照	○

3.2.5 イベントフラグ

ITRON の API 名称	機能	利用の可否と制限事項
CRE_FLG, cre_flg, acre_flg	イベントフラグの生成	○
del_flg	イベントフラグの削除	○
set_flg, iset_flg	イベントフラグのセット	○
clr_flg	イベントフラグのクリア	○
wai_flg, pol_flg, twai_flg	イベントフラグ待ち	○
ref_flg	イベントフラグの状態参照	○

3.2.6 データキュー

ITRON の API 名称	機能	利用の可否と制限事項
CRE_DTQ, cre_dtq, acre_dtq	データキューの生成	○ *6
del_dtq	データキューの削除	○
snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq	データキューへの送信	○
fsnd_dtq, ifsnd_dtq	データキューへの強制送信	× *7
rcv_dtq, prcv_dtq, trcv_dtq	データキューからの受信	○
ref_dtq	データキューの状態参照	○

*6 データキュー用のメモリ領域は、常に OS 側で確保しますので、データキュー生成時の領域の先頭番地 (dtq) の指定はサポートしません。常に NULL とみなされます。

*7 データキューへの強制送信はサポートしません。

3.2.7 メールボックス

ITRON の API 名称	機能	利用の可否と制限事項
CRE_MBX, cre_mbx, acre_mbx	メールボックスの生成	○ *8
del_mbx	メールボックスの削除	○
snd_mbx	メールボックスへの送信	○
rcv_mbx, prcv_mbx, trcv_mbx	メールボックスからの受信	○
ref_mbx	メールボックスの状態参照	○

*8 メッセージ優先度順 (TA_MPRI) の場合も、すべてのメッセージは、メッセージ優先度にかかわらず、同一のメッセージキューで管理されます。メールボックス生成時の maxmpri と mprihd の指定は意味を持ちません。

3.2.8 ミューテックス

ITRON の API 名称	機能	利用の可否と制限事項
CRE_MTX, cre_mtx, acre_mtx	ミューテックスの生成	○
del_mtx	ミューテックスの削除	○
loc_mtx, ploc_mtx, tloc_mtx	ミューテックスのロック	○
unl_mtx	ミューテックスのロック解除	○
ref_mtx	ミューテックスの状態参照	○

3.2.9 メッセージバッファ

ITRON の API 名称	機能	利用の可否と制限事項
CRE_MBF, cre_mbf, acre_mbf	メッセージバッファの生成	○ *9
del_mbf	メッセージバッファの削除	○
snd_mbf, psnd_mbf, tsnd_mbf	メッセージバッファへの送信	○
rcv_mbf, prcv_mbf, trcv_mbf	メッセージバッファからの受信	○
ref_mbf	メッセージバッファの状態参照	○ *10

*9 メッセージバッファ用のメモリ領域は、常に OS 側で確保しますので、メッセージバッファ生成時の領域の先頭番地 (mbf) の指定はサポートしません。常に NULL とみなされます。

*10 smsgcnt はメッセージの個数ではなく有無を表します。メッセージがなければ 0, メッセージがあれば 1 となります。

3.2.10 ランデブ

ITRON の API 名称	機能	利用の可否と制限事項
CRE_POR, cre_por, acre_por	ランデブポートの生成	○
del_por	ランデブポートの削除	○
cal_por, tcal_por	ランデブの呼出し	○ *11
acp_por, pacp_por, tacp_por	ランデブの受付	○
fwd_por	ランデブの回送	○
rpl_rdv	ランデブの終了	○
ref_por	ランデブポートの状態参照	○
ref_rdv	ランデブの状態参照	○

*11 tcal_por でのタイムアウト指定の意味は、ランデブ終了までの時間ではなくランデブ成立までの時間となります。

3.2.11 固定長メモリプール

ITRON の API 名称	機能	利用の可否と制限事項
CRE_MPF, cre_mpf, acre_mpf	固定長メモリプールの生成	○ *12
del_mpf	固定長メモリプールの削除	○
get_mpf, pget_mpf, tget_mpf	固定長メモリブロックの獲得	○
rel_mpf	固定長メモリブロックの返却	○
ref_mpf	固定長メモリプールの状態参照	○

*12 固定長メモリプール用の領域は、常に OS 側で確保しますので、固定長メモリプール生成時の領域の先頭番地 (mpf) の指定はサポートしません。常に NULL とみなされます。

3.2.12 可変長メモリプール

ITRON の API 名称	機能	利用の可否と制限事項
CRE_MPL, cre_mpl, acre_mpl	可変長メモリプールの生成	○ *13
del_mpl	可変長メモリプールの削除	○
get_mpl, pget_mpl, tget_mpl	可変長メモリブロックの獲得	○
rel_mpl	可変長メモリブロックの返却	○
ref_mpl	可変長メモリプールの状態参照	○

*13 可変長メモリプール用の領域は、常に OS 側で確保しますので、可変長メモリプール生成時の領域の先頭番地 (mpl) の指定はサポートしません。常に NULL とみなされます。

3.2.13 システム時刻管理

ITRON の API 名称	機能	利用の可否と制限事項
set_tim	システム時刻の設定	○
get_tim	システム時刻の参照	○
isig_tim	タイムティックの提供	× *14

*14 システム時刻は OS 側で自動更新するため、isig_tim は特に何もせず E_OK を返します。

3.2.14 周期ハンドラ

ITRON の API 名称	機能	利用の可否と制限事項
CRE_CYC, cre_cyc, acre_cyc	周期ハンドラの生成	○
del_cyc	周期ハンドラの削除	○
sta_cyc	周期ハンドラの動作開始	○
stp_cyc	周期ハンドラの動作停止	○
ref_cyc	周期ハンドラの状態参照	○

3.2.15 アラームハンドラ

ITRON の API 名称	機能	利用の可否と制限事項
CRE_ALM, cre_alm, acre_alm	アラームハンドラの生成	○
del_alm	アラームハンドラの削除	○
sta_alm	アラームハンドラの動作開始	○
stp_alm	アラームハンドラの動作停止	○
ref_alm	アラームハンドラの状態参照	○

3.2.16 オーバーランハンドラ

ITRON の API 名称	機能	利用の可否と制限事項
DEF_OVR, def_ovr	オーバーランハンドラの定義	○ *15
sta_ovr	オーバーランハンドラの動作開始	○
stp_ovr	オーバーランハンドラの動作停止	○
ref_ovr	オーバーランハンドラの状態参照	○

*15 オーバーランハンドラ属性は TA_HLNG のみ指定可能です。TA_ASM はサポートしません。

3.2.17 システム状態管理機能

ITRON の API 名称	機能	利用の可否と制限事項
rot_rdq, irot_rdq	タスクの優先順位の回転	○
get_tid, iget_tid	実行状態のタスク ID の参照	○
loc_cpu, iloc_cpu	CPU ロック状態への移行	○ *16
unl_cpu, iunl_cpu	CPU ロック状態の解除	○ *16
dis_dsp	ディスパッチの禁止	○
ena_dsp	ディスパッチの許可	○
sns_ctx	コンテキストの参照	○
sns_loc	CPU ロック状態の参照	○ *16
sns_dsp	ディスパッチ禁止状態の参照	○
sns_dpn	ディスパッチ保留状態の参照	○
ref_sys	システムの状態参照	○

*16 CPU ロック状態の意味は、ディスパッチ禁止状態かつ割込み禁止状態とします。

3.2.18 割込み管理機能

ITRON の API 名称	機能	利用の可否と制限事項
DEF_INH, def_inh	割込みハンドラの定義	○
ATT_ISR, cre_isr, acre_isr	割込みサービスルーチンの追加/生成	○ *17
del_isr	割込みサービスルーチンの削除	○
ref_isr	割込みサービスルーチンの状態参照	○
dis_int	割込みの禁止	○
ena_int	割込みの許可	○
chg_ixx, get_ixx	割込みマスクの変更/参照	× *18

*17 割込みサービスルーチン属性は TA_HLNG のみ指定可能です。TA_ASM はサポートしません。

*18 割込みマスクの変更/参照は ITRON の API としてはサポートしませんが、T-Kernel 側のライブラリを使って割込みの各種設定が可能です。

3.2.19 サービスコール管理機能

API 名	機能	利用の可否と制限事項
DEF_SVC, def_svc	拡張サービスコールの定義	× *19
cal_svc	サービスコールの呼出し	× *19

*19 サービスコール管理機能はサポートしません。

3.2.20 システム構成管理機能

ITRON の API 名称	機能	利用の可否と制限事項
DEF_EXC, def_exc	CPU 例外ハンドラの定義	× *20
ref_cfg	コンフィギュレーション情報の参照	○
ref_ver	バージョン情報の参照	○
ATT_INI	初期化ルーチンの追加	○ *21

*20 CPU 例外ハンドラはサポートしません。
*21 初期化ルーチン属性は TA_HLNG のみ指定可能です。TA_ASM はサポートしません。

3.3 実装定義事項

μITRON4.0 仕様で「実装毎に規定すべき事項」とされる主な事項について、I-right/TK でどのようになっているかを説明します。

3.3.1 データ型

μITRON4.0 仕様で使われる大文字表記のデータ型は、ヘッダファイル itron/itron.h をインクルードすることにより利用できます。μITRON4.0 仕様で幅の固定されていないデータ型 (INT 型、UINT 型、ER 型、ID 型など) については、T-Kernel に合わせていずれも 32 ビット以上の幅を持ちます。詳細は『T-Kernel 仕様書』をご参照ください。

システム時刻を表す SYSTIM 型は、T-Kernel と同じく上位 32 ビット (hi)、下位 32 ビット (lo) の構造体で、単位はミリ秒です。

3.3.2 オブジェクト ID 番号の範囲と最大個数

ITRON のタスクやセマフォなどのオブジェクト ID 番号の範囲は、最小値は 1 (固定) で、最大値は T-Kernel の SYSCONF (システム構成情報) の下記の設定と一致します。例えば SYSCONF で TMaxTskId が 150 に設定されている場合は、タスク ID は 1 から 150 までとなります。

TMaxTskId	タスク ID の最大値
TMaxSemId	セマフォ ID の最大値
TMaxFlgId	イベントフラグ ID の最大値
TMaxMbxId	メールボックス ID の最大値
TMaxMtxId	ミューテックス ID の最大値
TMaxMbfId	メッセージバッファ ID の最大値
TMaxPorId	ランデブポート ID の最大値
TMaxMpfId	固定長メモリプール ID の最大値
TMaxMplId	可変長メモリプール ID の最大値
TMaxCycId	周期ハンドラ ID の最大値
TMaxAlmId	アラームハンドラ ID の最大値

SYSCONF の各設定値の参照や変更については、μTeaboard/ARM7-AT91 の『実装仕様書』をご参照ください。

ただしデータキューに関しては、T-Kernel 側に直接対応する機能がないため、T-Kernel のメッセージバッファの範囲 (TMaxMbfId) を ITRON のデータキューとメッセージバッファで共通に使います。

ITRON オブジェクトの最大個数については、T-Kernel 側でもタスクやセマフォなどのオブジェクトを使用しているため、ITRON 側で使える個数はその分だけ少なくなります。例えば SYSCONF で TMaxTskId が 150 に設定されている場合、ITRON 側と T-Kernel 側でそれぞれ生成できるタスクの個数の合計が 150 となります。タスクの個数が不足する場合は、SYSCONF の TMaxTskId の設定を変更して最大値を増やしてください。

システム上で既に使われているオブジェクトの一覧は、次のコマンドで見ることができます。ただし、表示されるオブジェクト ID 番号の値は T-Kernel のオブジェクト ID 番号 (動的に割り当てられた ID 番号) であり、ITRON の API から見たオブジェクト ID 番号 (固定的に割り当てられた ID 番号) とは異なります。

- IMS のコマンドライン上から ref_tsk コマンド、ref_sem コマンドなど

3.3.3 システムタイマー

ITRON 側のシステムタイマーは、T-Kernel 側のシステムタイマーを利用します。システムタイマーの間隔は SYSCONF の TTimPeriod でミリ秒単位で設定でき、デフォルトでは 10 ミリ秒です。

ITRON 側のシステム時刻は、コンフィギュレーション終了 (END_CFG) の際に自動的にゼロに初期化されます。ただしこれは T-Kernel 側のシステム時刻には影響を与えません。

3.3.4 タスク優先度の範囲

ITRON タスクの優先度は、デフォルトでは 1 (最高) から 41 (最低) の範囲を指定できます。この範囲は T-Kernel タスクの優先度 100 から 140 に対応づけられます。

デフォルトの範囲	
ITRON タスク優先度	T-Kernel タスク優先度
対応なし	1
⋮	⋮
対応なし	99
1	100
⋮	⋮
41	140

ただしコンフィギュレーション開始 (`STA_CFG`) より前に `SET_CFG_TK_TPRI` を指定することにより、この対応を変更できます。例えば `SET_CFG_TK_TPRI(80)` とすれば、ITRON タスクの優先度 1 が T-Kernel タスクの優先度 80 に対応づけられ、全体として次のような対応になります。

`SET_CFG_TK_TPRI(80)` の場合	
ITRON タスク優先度	T-Kernel タスク優先度
対応なし	1
⋮	⋮
対応なし	79
1	80
⋮	⋮
61	140

3.3.5 割込みハンドラと割込みサービスルーチン

I-right/TK では割込みハンドラと割込みサービスルーチンの両方をサポートします。ただし同一の割込み番号に対しては、割込みハンドラと割込みサービスルーチンのどちらか一方のみ使用可能です。両方同時に使用することはできません。

- **割込みハンドラ**

  μITRON4.0 仕様では、割込みハンドラの詳細は実装定義となっています。I-right/TK では、割込みハンドラは T-Kernel の割込みハンドラと同じ形式です。詳細は機種ごとに異なります。詳しくは μTeaboard/ARM7-AT91 の『実装仕様書』をご参照ください。

- **割込みサービスルーチン**

割込みサービスルーチンは割込みハンドラを抽象化して機種依存性を少なくしたものです。I-right/TK においても μITRON4.0 仕様に従った割込みサービスルーチンを利用できます。

割込みサービスルーチンの ID の範囲は、1 (固定) から 20 (デフォルト) の範囲を指定できます。ただしコンフィギュレーション開始 (`STA_CFG`) より前に `SET_CFG_MAX_ISRID` を指定することにより、最大値を変更できます。例えば `SET_CFG_MAX_ISRID(30)` とすれば、割込みサービスルーチンの ID の範囲は 1 〜 30 になります。なお割込みサービスルーチンの最大個数は、静的 API の `ATT_ISR` で追加する分も含まれます。`cre_isr` や `acre_isr` で動的に生成できる個数はその分だけ少なくなります。

割込みハンドラの場合も割込みサービスルーチンの場合も、割込みの各種設定の詳細については I-right/TK の機能には含まれませんので、必要に応じて T-Kernel 側のシステムライブラリ (`<tk/syslib.h>`) の関数やマクロを呼び出すか、割込みコントローラを直接操作してください。詳しくは μTeaboard/ARM7-AT91 の『実装仕様書』をご参照ください。

同様にハードウェアの I/O ポート・レジスタの設定についても、システムライブラリの次のマクロを利用できます。

`in_b( アドレス )`	I/O の読込み (8 ビット幅)
`in_h( アドレス )`	I/O の読込み (16 ビット幅)
`in_w( アドレス )`	I/O の読込み (32 ビット幅)
`out_b( アドレス, 値 )`	I/O の書込み (8 ビット幅)
`out_h( アドレス, 値 )`	I/O の書込み (16 ビット幅)
`out_w( アドレス, 値 )`	I/O の書込み (32 ビット幅)

3.3.6 タスク固有空間の参照と設定

T-Kernel におけるメモリ空間は、全タスク共通のシステム共有空間の他に、タスク固有空間と呼ばれる空間をタスクごとに持つことができます。この機能を使って、アプリケーションプロセスはそれぞれ独立したメモリ空間上で動作します。(詳細につきましては『T-Kernel 仕様書』をご参照ください)

I-right/TK を使って ITRON でプログラムを作成する場合、プログラムそのものは全タスク共通のシステム共有空間上で動作します。しかし、プロセスとの間の通信を行う場合など、タスク固有空間を操作する必要が生じることがあります。このため I-right/TK では、μITRON4.0 仕様外の機能として以下の API も利用することができます。

タスク固有空間の参照

【C 言語 API】

```
ER ercd = vget_tsp( ID tskid, T_TSKSPC *pk_tskspc );
```

【パラメータ】

ID	tskid	対象タスクの ID

【リターンパラメータ】

ER	ercd	エラーコード
T_TSKSPC	tskspc	タスク固有空間情報

【機能】

tskid で指定したタスクのタスク固有空間を取得します。

T-Kernel の tk_get_tsp と同等の機能です。

タスク固有空間の設定

【C 言語 API】

```
ER ercd = vset_tsp( ID tskid, T_TSKSPC *pk_tskspc );
```

【パラメータ】

ID	tskid	対象タスクの ID
T_TSKSPC	tskspc	タスク固有空間情報

【リターンパラメータ】

ER	ercd	エラーコード

【機能】

tskid で指定したタスクのタスク固有空間を設定します。

T-Kernel の tk_set_tsp と同等の機能です。

4 T-Kernel 側の機能との連携

I-right/TK を使って動作する ITRON のプログラムから T-Kernel 側のライブラリやミドルウェアを呼び出したり、他プログラムと連携動作させる方法についてまとめます。

これらの機能を使った具体的なプログラム例としては、以下のサンプルが付属していますので、あわせてご参照いただければ幸いです。

`sample_lib`	ライブラリを使うサンプル
`sample_net`	ネットワーク (TCP/IP) にアクセスするサンプル
`sample_dev`	デバイスドライバを呼び出すサンプル
`sample_sdi`	デバイスドライバを作成して他のプログラムと連携動作するサンプル

なお、本章で説明のない機能 (通常の T-Kernel のシステムコールや、T-Kernel Extension の機能など) については、ITRON のプログラムから直接呼び出して利用することはできません。逆に、T-Kernel のタスクや T-Kernel Extension のプロセスから ITRON のプログラムを呼び出すことは、ITRON のプログラムを T-Kernel のデバイスドライバとして登録することによって可能です。この方法の詳細につきましては、4.4 節をご覧ください。

4.1 ライブラリ

μTeaboard/ARM7-AT91 用開発環境に付属するライブラリは、I-right/TK を使って動作する ITRON のプログラムからも基本的に利用可能です。特に C 言語標準ライブラリである `stdio.h`、`stdlib.h`、`string.h` なども利用できます。ライブラリの詳細につきましては μTeaboard/ARM7-AT91 の『ライブラリ説明書』をご参照ください。

I-right/TK を使って動作する ITRON のプログラムからライブラリを使う場合の注意事項は次のとおりです。

- **ヘッダファイルのインクルード**

  ヘッダーファイルをインクルードする場合は、`itron/itron.h` より後に、ライブラリなど他のヘッダーファイルをインクルードする必要があります。`itron/itron.h` より先に他のヘッダーファイルをインクルードすると、問題を生じる可能性があります。

- **非タスクコンテキストでは一部の機能が利用不可**

  `stdio.h` の `printf()` など、待ち状態やデバイスアクセスを伴うライブラリ関数の場合は、タスクからのみ利用できます。割込みハンドラ等の非タスクコンテキストからは利用できません。

- **プロセス専用のライブラリは利用不可**

  ITRON のプログラムから利用可能なライブラリは、T-Kernel ベースのプログラムから利用可能なライブラリに限られます。ダイナミックリンクライブラリ (`btron/dynload.h`) など、プロセス専用のライブラリは利用できません。

- **常駐メモリと非常駐メモリ**

  `stdlib.h` の `malloc()` 等で確保されるメモリは非常駐メモリ (仮想記憶で管理されるメモリ) です。非常駐メモリは通常のアプリケーションからアクセスする場合は問題ありませんが、割込み禁止状態ではアクセスできず、また非常駐メモリのアドレスを ITRON の API に渡すことも、一般にはできません。たとえば、メッセージバッファへの送信 (`snd_mbf`) の際、送信メッセージ先頭番地 (`msg`) として非常駐メモリのアドレスを渡すことはできません。ITRON の API のパラメータとして渡せるメモリアドレスは、常駐メモリ (実記憶で管理されるメモリ) のアドレスである必要があります。常駐メモリを確保するには `tk/sysmgr.h` (T-Kernel/SM) の `Kmalloc()` 等をご利用ください。

4.2 ネットワーク (TCP/IP)

I-right/TK を使って動作する ITRON のプログラムから、TCP/IP マネージャの API を使ってネットワーク (TCP/IP) へのアクセスが可能です。インクルードファイルは `btron/bsocket.h` です。

この場合の注意事項は次のとおりです。

- **ヘッダファイルのインクルード**

  ヘッダーファイル `btron/bsocket.h` は `itron/itron.h` より後でインクルードする必要があります。`itron/itron.h` より先に他のヘッダーファイルをインクルードすると、問題を生じる可能性があります。

- **`so_start` でネットワークを開始する**

  ネットワークを開始するには `so_start` を呼び出す必要があります。

- **API 名の頭に `so_` をつける**

  `so_socket`、`so_connect`、`so_send`、`so_recv`、`so_close` のように、API 名の頭に `so_` をつける必要があります。この点を除けば、T-Kernel 用の TCP/IP マネージャの API は、標準的なソケット API にほぼ準拠しています。TCP/IP マネージャの API の仕様の詳細については、『TCP/IP マネージャ説明書』をご参照ください。

- **非タスクコンテキストでは利用不可**

  TCP/IP の機能はタスクからのみ利用できます。割込みハンドラ等の非タスクコンテキストからは利用できません。

4.3 デバイスドライバの呼出し

I-right/TK を使って動作する ITRON のプログラムから、T-Kernel のデバイス管理の API (`tk_opn_dev` など) を使ってデバイスドライバの呼出しが可能です。インクルードファイルは `tk/devmgr.h` です。なお T-Kernel のデバイスドライバの詳細については『T-Kernel 仕様書』をご参照ください。

この場合の注意事項は次のとおりです。

- **ヘッダファイルのインクルード**

  `tk/devmgr.h` や各デバイスドライバのヘッダファイルは `itron/itron.h` より後でインクルードする必要があります。`itron/itron.h` より先に他のヘッダーファイルをインクルードすると、問題を生じる可能性があります。

- **非タスクコンテキストでは利用不可**

  デバイスドライバはタスクからのみ呼び出せます。割込みハンドラ等の非タスクコンテキストからは利用できません。

4.4 ITRON プログラムのデバイスドライバ化による連携動作

I-right/TK を使って動作する ITRON のプログラムと、他のプログラム (T-Kernel ベースのプログラムなど) の間の連携を取る方法の一つとして、ITRON 側のプログラムをデバイスドライバとして登録する方法があります。

デバイスドライバとして登録するには、デバイスドライバインタフェース層というライブラリを用います。具体的には次の SDI または GDI のどちらかを利用目的に応じて選択してご使用ください。なお、これらの詳細につきましては、μTeaboard/ARM7-AT91 の『デバイスドライバ説明書』の「デバイスドライバ I/F 層」をご参照ください。

- **単純デバイスドライバインタフェース層 (SDI)**

  すべての処理中に不定期の待ちが発生せず、すみやかに処理が終了する場合に使います。ヘッダファイルは `device/sdrvif.h` です。

  デバイスドライバ内の処理は、呼び出し側と同じコンテキストで実行される関数内で行われます。

- **汎用デバイスドライバインタフェース層 (GDI)**

  一般的な処理、特に処理中に不定期の待ちが発生する場合に使います。ヘッダファイルは `device/gdrvif.h` です。

  デバイスドライバ内の処理は、デバイスドライバ内で独自に作成したタスクで行います。

この場合の注意事項は次のとおりです。

- **ヘッダファイルのインクルード**

  `device/sdrvif.h` や `device/gdrvif.h` は `itron/itron.h` より後でインクルードする必要があります。`itron/itron.h` より先に他のヘッダーファイルをインクルードすると、問題を生じる可能性があります。

- **非タスクコンテキストではドライバインタフェース層の関数は利用不可**

  『デバイスドライバ説明書』で明記された場合を除いて、非タスクコンテキストからはドライバインタフェース層の関数は利用できません。

- **常駐メモリと非常駐メモリ**

  プロセス内で通常の方法で確保されたメモリは非常駐メモリです。プロセスからそのメモリ領域を指定してデバイスリードやデバイスライトを発行した場合、デバイスドライバ側のタスクコンテキストからその非常駐メモリにアクセスすることは問題ありませんが、その非常駐メモリのアドレスを ITRON の API に渡すことはできません。たとえば、メッセージバッファへの送信 (`snd_mbf`) の際、送信メッセージ先頭番地 (`msg`) として非常駐メモリのアドレスを渡すことはできません。ITRON の API のパラメータとして渡せるメモリアドレスは、常駐メモリ (実記憶で管理されるメモリ) のアドレスである必要があります。

- **タスク固有空間の設定**

  デバイスドライバ内で独自に生成したタスク内で、呼出し側である他のプログラムから渡されたバッファにアクセスする場合、そのバッファがタスク固有空間上に置かれている可能性があります。また、必ずしもタスク固有空間が一致しているとは限りません。このため `vset_tsp` を使ってタスク固有空間を適切に設定する必要があります。

  なお、SDI のリード関数やライト関数のように、呼出し側と同じコンテキストで実行される関数については、タスク固有空間がもともと一致していますので、`vset_tsp` による設定は必要ありません。

索 引

I-right/TK 取扱説明書
(μTeaboard/ARM7-AT91 用)

パーソナルメディア株式会社
Web: http://www.t-engine4u.com/
E-Mail: te-sales@personal-media.co.jp